CS3723 Programming Assignment #4 - LISP Natural Language for DB Queries (50 pts) – due date – 2018-11-12

In this assignment, you will use LISP to understand natural language requests which represent database queries. Your code will parse the request into the parts of speech. For extra credit, you will generate a SQL statement for the parsed request.

If you use a solution from another webiste in this program, you will receive a 0 on this assignment, fail this course, and possibly be expelled from UTSA.

Sample Requests:

1. Count the number of rentals
2. Show ID for movies
3. Show title of movies with rating equal to PG13
4. Count the number of rentals with date after 2018-10-01
5. Show title for the movie with ID equal to HP001
6. Show title for movies with rating not equal to G
7. Show title for movies with genre equal to romance
8. Show title for movies in the horror genre (extra credit)
9. Count the number of movies with a PG13 rating (extra credit)

Simple request grammar:

Request := Verb DirectObj Prep IndirectObj [Prep QualifyingPhrase]
*QualifyingPhrase:= QualIndirectObj ['*not'*] Comparator [Prep] QualValue*
DirectObj := NounPhrase
IndirectObj := NounPhrase
QualIndirectObj := NounPhrase
NounPhrase := [Article] Noun

Notes:

1. Brackets indicate something that is optional. For example, an *Article* is optional in a *NounPhrase*.
2. The italicized words in black are nonterminal symbols, meaning that they are replaceable with something else.
3. A symbol in black surrounded by quotes is an actual word (e.g., 'not').
4. The symbols in blue within the grammar refer to words. Most of these are defined using the **`set_isa`** macro which you must code. (See the **`set_isa`** documentation below.) Additionally, you can determine the part of speech for a word by using the **`isa`** function (which you must code).
   - *Verb* values include at least count, select, show, and print.
   - *Prep* values include at least of, with, to, in, and for.
   - *Comparator* values include equal, after, and before.
   - *Noun* values include at least movies, movie, rentals, rental, customer, and customers. It will also include attribute names in the database tables.
   - *QualValue* includes too many values to list since it will reference database values. You will not validate these.
5. This program uses three hash tables:
   - parse-obj – contains the original request, a cursor position, and a value for each part of speech
   - word-dict – for most words (except *QualValue* words), this provides the part of speech
   - table-ht – this provides the actual table names. It resolves issues like plurals. For example, we want to understand that both "movie" and "movies" refer to the same database table "movie". There would be two different entries in the table-hr hash table:
     - movie → movie
     - movies → movie
6. For extra credit, the grammar will be more complicated. (See below.)
7. Turn in a zip file named LastNameFirstName.zip (no spaces) containing:

- **p4Lisp.txt** – your LISP source code.
- Your log of the session (see the setup instructions). This should be a **p4Out.txt**.
- **Do not have any directories within your zip file.**

Sample Results (partial):

```
;;; first form:  verb [article] directObj prep [article] indirectObj
 (processRequest '(count the number of rentals ))
 ****************************************************************
  (COUNT THE NUMBER OF RENTALS)
    checkRequest returned T
    verb= COUNT
    directObj= NUMBER
    prep= OF
    indirectObj= RENTALS
    DBRequest= (SELECT count(*) FROM RENTAL)
T
 (processRequest '(show ID for movies))
 ****************************************************************
  (SHOW ID FOR MOVIES)
    checkRequest returned T
    verb= SHOW
    directObj= ID
    prep= FOR
    indirectObj= MOVIES
    DBRequest= (SELECT ID FROM MOVIE)
T

 ;;; second form: verb [article] directObj prep [article] indirectObj
 ;;;                prep [article] qualIndirectObj comparator [prep] qualValue
 (processRequest '(show the title of movies with rating equal to PG13))
 ****************************************************************
  (SHOW THE TITLE OF MOVIES WITH RATING EQUAL TO PG13)
    checkRequest returned T
    verb= SHOW
    directObj= TITLE
    prep= OF
    indirectObj= MOVIES
    QualIndirectPrep= WITH
    QualIndirectObj= RATING
    Comparator= EQUAL
    QualPrep= TO
    QualValue= PG13
    DBRequest= (SELECT TITLE FROM MOVIE WHERE RATING = "PG13")
T
 (processRequest '(Count the number of rentals with date after 2018-10-01))
 ****************************************************************
  (COUNT THE NUMBER OF RENTALS WITH DATE AFTER 2018-10-01)
    checkRequest returned T
    verb= COUNT
    directObj= NUMBER
    prep= OF
    indirectObj= RENTALS
    QualIndirectPrep= WITH
    QualIndirectObj= DATE
    Comparator= AFTER
    QualPrep= NIL
    QualValue= 2018-10-01
    DBRequest= (SELECT count(*) FROM RENTAL WHERE DATE > "2018-10-01")
T
;;;  Examples with NOT
 (processRequest '(Show title for movies with rating not equal to G))
 ****************************************************************
  (SHOW TITLE FOR MOVIES WITH RATING NOT EQUAL TO G)
    checkRequest returned T
    verb= SHOW
```

```
    directObj= TITLE
    prep= FOR
    indirectObj= MOVIES
    QualIndirectPrep= WITH
    QualIndirectObj= RATING
    NotQual = NOT
    Comparator= EQUAL
    QualPrep= TO
    QualValue= G
    DBRequest= (SELECT TITLE FROM MOVIE WHERE RATING <> "G")
T
 ;;; extra credit #2
 (processRequest '(Show the title of movies in the horror genre))
 ****************************************************************
  (SHOW THE TITLE OF MOVIES IN THE HORROR GENRE)
    checkRequest returned T
    verb= SHOW
    directObj= TITLE
    prep= OF
    indirectObj= MOVIES
    QualIndirectPrep= IN
    QualIndirectObj= GENRE
    Comparator= EQUAL
    QualPrep= TO
    QualValue= HORROR
    DBRequest= (SELECT TITLE FROM MOVIE WHERE GENRE = "HORROR")
T
 (processRequest '(Count the number of movies with a PG13 rating))
 ****************************************************************
  (COUNT THE NUMBER OF MOVIES WITH A PG13 RATING)
    checkRequest returned T
    verb= COUNT
    directObj= NUMBER
    prep= OF
    indirectObj= MOVIES
    QualIndirectPrep= WITH
    QualIndirectObj= RATING
    Comparator= EQUAL
    QualPrep= TO
    QualValue= PG13
    DBRequest= (SELECT count(*) FROM MOVIE WHERE RATING = "PG13")T
```

Parsing a Request

There are many parsing approaches; however, you are **required** to use the approach described here.

- You have been provided a parse-obj which is simply a hash table. It contains the request to parse, current cursor position (relative to zero), and each part of speech (e.g., subject, verb, directObj, prep, indirectObj).
- You have been provided a word dictionary which is simply a hash table. Each word has a single identified part of speech. The words will be assigned a part of speech using the **set_isa** macro. To obtain a word's part of speech, use the **isa** function. The word is the key to the hash-table.
- To access the next word in a request, you have been provided a **getToken** function which increments the current cursor position and returns the current token prior to that position change. getToken **must** be used to advance through a request. getToken uses **getCursor** (provided) and **setCursor** (provided).
- Example using **getCursor** and **isa** (partial code):

```
(defun checkStuff (parse objNm)
    (prog (preposition) ;;; local variable named prep
         ;;; get a preposition
         (setf preposition (getToken parse))
         ;;; it must be a preposition
         (if (not (isa preposition 'prep))
             (return NIL) )
         (if (not (checkNP parse objNm))
             (return NIL) )
```

…

- One difficulty in parsing is when it is necessary to "back up". It is likely that the extra credit will need to "back up". In our parsing approach if it is necessary to "back up", save the cursor position before parsing some portion and then use **setCursor** to set to the old position.
  Example (assume parse is set to parse-obj)

```
;;; Assume checkNP uses getToken to get tokens.
;;; If the surrounding code needs to reset the position, it is
;;; done by simply calling setCursor as shown
(setf saveCursor (getCursor parse)) ;;; save current cursor position
(if (not (checkNP parse 'directObj))
    (setCursor parse saveCursor))   ;;; back up to the old position
```

- When saving the parts of speech, which is how you provide results, use (putp *partOfSpeech* parse *value*)
- **You** are **required** to implement the following functions:

(**checkRequest** *parse*)	This function checks for a valid request according to the grammar above. Assume it is passed a parse-obj (which has already been populated with a request) and returns **T** if the request is valid; otherwise, it returns **NIL**. Name the parameter **parse** to avoid conflicts. Additionally, checkRequest **saves** the **identified parts of speech**. checkRequest is invoked by Larry's **processRequest** (see the sample output).
(**resetPartsOfSpeech** *parse partOfSpeech1 partOfSpeech2 ...*)	This function resets the value for each of the specified parts of speech to NIL using **putp**. The first argument is a parse-obj. There are a variable number of parts of speech passed to resetPartsOfSpeech. Example: `(resetPartsOfSpeech parse-obj 'subject 'verb 'prep 'directObj)`

(**set_isa** *partOfSpeech word1 word2 …*)	This macro defines each word in the list of *words* to the specified *partOfSpeech* in the dictionary (hard code word-dict). Use (**putp** *word* **word-dict** *partOfSpeech***)** to put each word in word-dict with the *partOfSpeech.*
	Examples:
	`(set_isa article a an the)`
	`(set_isa verb count select show print)`

(**isa** *word partOfSpeech*)	returns T if the specified *word* is that specified *partOfSpeech*; otherwise, NIL is returned. Example: `(isa 'show 'verb)` returns T based on the set_isa above.

Larry provided

- p4LispDef.txt – some LISP functions which you need to load prior to loading your p4Lisp.txt
- p4LispRun.txt – this will execute your code and should be loaded after you load your code

Functions Larry Provided (see p4LispDef.txt for more information)

(processRequest *request*) - invokes your functions to process a request. It invokes resetPartsOfSpeech and checkRequest. You won't use this function directly; instead, p4LispRun.txt uses it.
(putp *symbol ht value*) - puts a property about a symbol into a hash table
(getp *symbol ht*) - gets a property about a symbol from a hash table
(getCursor *parse*) - returns the current cursor position for the specified parse
(setCursor *parse cursorPosition*) - sets the current cursor position (usually with a previously saved getCursor position)
(getToken *parse*) - gets the next token from the request. It also advances the cursor position.

Extra Credit:

Inside **p4Lisp.txt**, setf **doingExtra** to one of these values:

- **NIL**

- **'EC1**
- **'EC2**

To receive extra credit, your submission must not be late and it must meet all requirements.

EC#1: (10+200pts/n) Generate SQL select statements.

- processRequest invokes (genSQL *parse*) to generate and return a list containing a well formed SQL statement.
- If the verb is COUNT, it generates SELECT COUNT(*). Otherwise it generates SELECT *directObj*
- The FROM tableName uses table-ht to generate the correct table name.
- If there is a *QualifyingPhrase*, you will need to generate a WHERE clause.
    - Examples:
        1. Parts of Speech:
           ```
           QualIndirectObj= RATING
           Comparator= EQUAL
           QualPrep= TO
           QualValue= PG13
           ```
           WHERE clause:
           ```
           WHERE RATING = "PG13"
           ```
        2. Parts of Speech:
           ```
           QualIndirectObj= DATE
           Comparator= AFTER
           QualPrep= NIL
           QualValue= 2018-10-01
           WHERE clause:
           WHERE DATE > "2018-10-01"
           ```
        3. Parts of Speech:
           ```
           QualIndirectObj= RATING
           NotQual = NOT
           Comparator= EQUAL
           QualPrep= TO
           QualValue= G
           WHERE clause:
           WHERE RATING <> "G"
           ```
    - To surround the value of, qualValue, with double quotes, use:
      `(format NIL "\"~a\"" qualValue)`
      If `qualValue` was `'PG13,` this call of `format` would return `"PG13".`

EC#2: (5+100pts/n) Determine *Comparator [Prep] QualValue* based on *QualifyingPhrase* containing *[Article] Adjective Noun*

- Must also meet all requriements of EC#1
- Grammar:

  Request := Verb DirectObj Prep IndirectObj [Prep QualifyingPhrase]
  QualifyingPhrase:= QualIndirectObj [not] *Comparator [Prep] QualValue*
  | *[Article] Adjective Noun*
  DirectObj := NounPhrase
  IndirectObj := NounPhrase
  QualIndirectObj := NounPhrase
  NounPhrase := [Article] Noun

Examples:

- Show the title of movies in the horror genre
    - Initially, we recognize it as
      Verb: show
      DirectObj: title
      Prep: of

IndirectObj: movies
QualIndirectPrep: in
QualIndirectObj: genre
QualVerb: NIL
QualPrep: NIL
Adjective: horror

- We then map it to
  Verb: show
  DirectObj: title
  Prep: of
  IndirectObj: movies
  QualIndirectPrep: in
  QualIndirectObj: genre
  QualVerb: equal
  QualPrep: to
  QualValue: horror

Extra credit notes:

1. n is the total number of people from all sections who meet **all the requirements** for that particular extra credit.
2. Your submission **must not be late**.
3. Your code must meet all the LISP programming standards.
4. Your code must be properly documented and successfully handle ALL the test cases in p4LispRun.txt